SECURITY CLASSIFICATION OF THIS PAGE

AD-A200 741

DTIC FILE COPY

DTIC ELECTE OCT 13 1988 D

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 88-1117
6a. NAME OF PERFORMING ORGANIZATION University of Illinois	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State, and ZIP Code)	7a. NAME OF MONITORING ORGANIZATION Directorate of Mathematical and Information Science, AFOSR
	7b. ADDRESS (City, State, and ZIP Code) Building 410 Bolling AFB, DC 20332-6448
8a. NAME OF FUNDING / SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-85-0211
8c. ADDRESS (City, State, and ZIP Code) Building 410 Bolling AFB, DC 20332-6448	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
6112F	2304	A5	

11. TITLE (Include Security Classification)
Solving Linear Systems on Multiprocessors

12. PERSONAL AUTHOR(S)
Ahmed H. Sameh

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Report	FROM Aug 88 TO Aug 88		

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this research was to study robust iterative methods for solving sparse (block tridiagonal) nonsymmetric linear systems in a parallel computing environment. A new method was developed which uses block-row symmetric successive overrelaxation (SSOR) with conjugate gradient (CG) acceleration. The method is reobust, with convergence assured even for poorly conditioned systems, and the method is easily implemented in a parallel environment. The method transforms a nonsymmetric system with an arbitrary eigenvalue distribution into a symmetric one with eigenvalue restricted to the interval (0,1). Research included testing of the algorithms on an Alliant FX/8 multiprocessor where it was demonstrated that the methods is very robust and performs better than standard existing methods.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Lt Col D. Nelson	(202) 767-5026	NM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

88 1011 152

Final Report

AFOSR-85-0211

Title: Solving Linear Systems on Multiprocessors

Principal Investigator: Professor Ahmed H. Sameh

Institution: Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign

A Robust Parallel Iterative Solver for Sparse Nonsymmetric Linear Systems

A. Introduction

Block tridiagonal linear systems occur frequently in scientific and engineering computations, often from a linearization and discretisation of a nonlinear differential operator. In this setting three features are common: many such systems must be solved, each system is large–scale, and the coefficient matrix is possibly nonsymmetric. The first feature implies that an iterative scheme is desirable since it allows the linear systems to be solved only to the required accuracy, which needs to be high only for the final few systems. The second feature implies that an iterative scheme may be necessary, since factorisation of a large linear system may exceed the available computer memory. The third feature rules out several highly successful algorithms for symmetric positive definite systems, such as the conjugate gradient (CG) method.

This proposal describes a method developed in [KaSa86], funded by AFOSR, for the solution of nonsymmetric block tridiagonal systems, presents test results and outlines future research directions for developing a robust parallel iterative solver for nonsymmetric linear systems. The method, block–SSOR with CG acceleration, has two important properties. Firstly, it is robust. Convergence is assured even for poorly conditioned systems with arbitrary eigenvalue distributions. Secondly, the method can be implemented with good parallelism, making it suitable for multiprocessor machines.

Many current iterative solvers for nonsymmetric linear systems do not possess the first of these desirable properties. Chebyshev and related least–squares polynomial methods can fail if the origin is contained in the convex hull of the eigenvalues; in addition some estimate of the extremal eigenvalues is required. Generalisations of the conjugate gradient method such as GCR(k) and ORTHOMIN(k) require that the matrix has a positive definite symmetric part. GMRES(k) does not place any restriction on the matrix, but can require retaining a large number k of search directions and hence large amounts of additional storage in order to achieve convergence. Finally, iterative solvers applied to the normal equations can cause a loss of information as well as a squaring of the condition number, a potential source of numerical instability.

Section 2 of this proposal outlines the projection method under consideration. Section 3 discusses some implementation variations that make it suitable for a multiprocessor. Section 4 presents numerical results from tests run on an Alliant FX/8 minisupercomputer, and section 5 summarises conclusions about the algorithm, and suggests extensions for the general sparse case and other applications.

B. Description of the Algorithm

Given the nonsingular system $Ax = f$ with $A \in R^{N\times N}$ and $f \in R^N$, let the rows of A be partitioned as

$$A = \begin{bmatrix} B_1^T \\ B_2^T \end{bmatrix}, \tag{2.1}$$

where $B_1 \in R^{N\times M}$ and $B_2 \in R^{N\times(N-M)}$. Also, let f be partitioned conformally with A as $f = (f_1^T, f_2^T)^T$. Then the block–row symmetric successive over–relaxation (SSOR) method [BjEl79] with respect to this partitioning is defined as

Algorithm 1 (Block–row SSOR)

(a) Choose x_0 and $\omega \in (0,2)$. Set $k = 0$.

(b) Let

$$z_1 = x_k + \omega B_1 (B_1^T B_1)^{-1} (f_1 - B_1^T x_k)$$

$$z_2 = z_1 + \omega B_2 (B_2^T B_2)^{-1} (f_2 - B_2^T z_1)$$

$$z_3 = z_2 + \omega B_2 (B_2^T B_2)^{-1} (f_2 - B_2^T z_2)$$

$$x_{k+1} = z_3 + \omega B_1 (B_1^T B_1)^{-1} (f_1 - B_1^T z_3)$$

(c) If a stopping condition is not met, set $k = k+1$ and go to (b)

In [KaSa86] it was shown that the optimal relaxation parameter ω is 1; in this case the matrices $\omega B_i (B_i^T B_i)^{-1} B_i^T$ become orthogonal projectors and the algorithm can be simplified:

Algorithm 2 (Optimal block–row SSOR)

(a) Choose x_0 and set $k = 0$.

(b) Set $x_{k+1} = Q x_k + \tilde{f}$.

(c) If a stopping condition is not met set $k = k+1$ and go to (b).

Here the iteration matrix is defined as

$$Q = (I - P_1)(I - P_2)(I - P_1),$$

where

$$I - P_i = I - B_i (B_i^T B_i)^{-1} B_i^T$$

$$= \text{projector onto nullspace of } B_i^T$$

and the modified right–hand side vector is

$$\tilde{f} = [(I + (I - P_1)(I - P_2))(B_1^+)^T , (I - P_1)(B_2^+)^T] f,$$

with

$$B_i^+ := (B_i^T B_i)^{-1} B_i^T$$

Because Q is a product of orthogonal projectors it is easy to show that Q is symmetric and that the spectral radius $\rho(Q)$ is bounded above by 1. Furthermore, since A is nonsingular $\rho(Q) < 1$. Unfortunately $\rho(Q)$ depends on $\cos^2\theta$, where θ is the smallest angle between the two subspaces $S_1 = \text{null}(B_1^T)$ and $S_2 = \text{null}(B_2^T)$ (see [BjGo73] for a definition of θ). This angle can be small, with a correspondingly slow rate of convergence for Algorithm 2.

Therefore some acceleration technique is desirable. Since $\rho(Q) < 1$ implies that $(I - Q)$ is positive definite, the conjugate gradient method is a natural choice. Using a version of CG from [Reid71] gives the algorithm presented in [KaSa86] and that this paper further examines:

Algorithm 3 (Block SSOR with CG acceleration)

(a) Initialize $x_0 = 0$, $k = 0$, $p_0 = r_0 = \tilde{f}$, and $\rho_0 = r_0^T r_0$.

(b) Set

$$w_k = (I - Q) p_k$$

$$\alpha_k = \rho_k / p_k^T w_k$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k w_k$$

$$\rho_{k+1} = r_{k+1}^T r_{k+1}$$

$$\beta_{k+1} = \rho_{k+1} / \rho_k$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

(c) If a stopping condition is not met, set $k = k+1$ and go to (b).

This last method will be called BSSOR in the remaining sections, with only one modification. If x_0 is chosen to be the vector $\tilde{f}$, then $P_1 r_0 = P_1 p_0 = 0$. In this case

$$w_0 = [I - (I-P_1)(I-P_2)(I-P_1)]\, p_0$$

$$= p_0 - (I-P_1)(I-P_2)\, p_0$$

$$= (I-P_1)\, p_0 - (I-P_1)(I-P_2)\, p_0$$

$$= (I-P_1) P_2 p_0$$

also satisfies $P_1 w_0 = 0$. Since r_{k+1} is a linear combination of r_k and w_k and p_{k+1} is a linear combination of p_k and r_{k+1}, an induction argument shows that $P_1 r_k = P_1 p_k = 0$ for all k. As above, w_k can then be computed using only two rather than three projections per iteration. This reduces execution times by 20% - 25%. This initialization also allows the updating of the true residual on each step, avoiding the need for an additional multiplication by A. The true residual can be updated by subtracting $\alpha_k A p_k$ on each step. Since $P_1 p_k = 0$, $B_1^T p_k = 0$ and those components of the true residual remain 0. Since $B_2^T p_k$ is computed as the first step in finding w_k, the corresponding components of the true residual can be updated by a saxpy operation. This leads to a 5% reduction in the overall execution time.

C. Implementations

C.1. Partitioning Choices

On each iteration BSSOR requires the formation of the product

$$w = (I-P_1) P_2$$

and hence the solution of two linear least squares problems. This seems to be an impractical way to solve linear systems because when A is dense solving a single least squares problem is generally more difficult than solving the original system. However, when A is block tridiagonal, that is, when $A = [E_i, T_i, D_i]$ where each E_i, T_i, and D_i is an $n \times n$ matrix, it is possible to select partitionings (2.1) that allow each projection P_i to be computed as a simultaneous set of smaller least squares subproblems. These independent subproblems can then be solved in parallel.

To illustrate this idea, consider the case when A has 24 block rows, and label them as 1 through 24. Then by putting block rows 1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, 22 into B_1^T and the others into B_2^T, both will have 6 completely separate larger blocks. This will be denoted by

$$B_1^T : (1,2),(5,6),(9,10),(13,14),(17,18),(21,22)$$

$$B_2^T : (3,4),(7,8),(11,12),(15,16),(19,20),(23,24),$$

where the parenthesis indicate the separation between the larger blocks.

This is not the only partition that gives disjoint blocks for the two projections. The number of rows put into each larger block can be varied, and it is not necessary for B_1^T and B_2^T to have the same size of blocks. Furthermore some rows can be allowed to appear more than once, e.g., row 3 can be put into one larger block of B_1^T and one of B_2^T.

Although tests are restricted to block tridiagonal systems in this paper, the only property that is important here is that the system can be row permuted to have separate blocks in each B_i^T. In particular, unstructured sparse systems generally have this property. Block tridiagonal systems are used here because they allow an *a priori* determination of suitable partitionings.

C.2. Solving the least squares problems

While all of the above versions allow parallel computations, it is still necessary to simultaneously solve least square subproblems of the form

$$\min_s \|w - Cs\|, \tag{3.1}$$

where C has block dimensions 4×2 for the above partition, and w consists of the components of $\bar{f}$ corresponding to C's columns.

Several algorithms exist for solving problem (3.1). Orthogonal factorisation techniques [GoVL83] have the advantage of being numerically stable, but require a relatively large amount of storage for the factors. Paige and Saunder's algorithm LSQR [PaSa82] is an iterative method which solves the augmented system

$$\begin{bmatrix} I & C \\ C^T & 0 \end{bmatrix} \begin{bmatrix} r \\ s \end{bmatrix} = \begin{bmatrix} w \\ 0 \end{bmatrix}; \tag{3.2}$$

it proved to be rather time consuming.

Another possible approach is to solve the normal equations

$$C^T C s = C^T w. \tag{3.3}$$

While it can be argued that this is no improvement over solving the normal equations for the original system $Ax = f$, test results of section 4 show that in practice the subproblems (3.3) remain well–conditioned even when the original system is ill–conditioned. Combined with the gain due to parallelism, this makes (3.3) a reasonable approach. Furthermore $C^T C$ is symmetric positive definite and so more efficient solvers are available.

The discussion above is not meant to be exhaustive but rather to reveal the issues involved. The solution method chosen for (3.1) or (3.3) will depend on the structure of C which in turn will depend on the source of the matrix A. An inexpensive iterative method with an early termination criterion could be used during the first few iterations of BSSOR, with a direct method subsequently used when the solution is near and high accuracy of the inner iterations is needed. This flexibility of the algorithm allows it to be custom–tailored to the problem being solved.

D. Numerical Results

D.1. Test Problems

BSSOR is tested with block tridiagonal systems created by using 5–point central difference operators corresponding to elliptic partial differential equations with Dirichlet boundary conditions on the unit square. The equations are the same set used in [Kama86], where a uniform grid of size $h = 1/(n+1)$ is used for both the x and y coordinates with n ranging between 8 and 64. Note that in this case n is both the block size and the number of blocks, so that A is of order n^2. The right–hand side function g is selected to give a predetermined solution so that both the residual norm and error norm can be checked. The test problems and their solutions follow:

Problem 1:

$$-u_{xx} - u_{yy} = g$$

$$u = x + y$$

Problem 2:

$$-u_{xx} - [(1+xy)u_y]_y - \beta[\cos(x)u_x + (e^{-x}+x)u_y] + 3u = g$$

$$\beta = 5, 250, 10000$$

$$u = x + y$$

Problem 3:

$$-(e^{-xy}u_x)_x - (e^{xy}u_y)_y + \beta(x+y)u_y + [\beta(x+y)u]_y + (1+x+y)^{-1}u = g$$

$$\beta = 5, 250, 10000$$

$$u = xe^{xy}\sin(\pi x)\sin(\pi y)$$

Problem 4:

$$-u_{xx} - u_{yy} - 100xu_x + yu_y - 100\frac{(x+y)}{xy}u = g$$

$$u = x + y$$

Problem 5:

$$-u_{xx} - u_{yy} - xu_x + 200yu_y - 300u = g$$

$$u = x + y$$

Problem 6:

$$-u_{xx} - u_{yy} + 1000e^{xy}(u_x - u_y) = g$$

$$u = x + y$$

In the testing 2a, 2b, 2c and 3a, 3b, 3c are used to differentiate between the values $\beta = 5, 250, 1000$, respectively.

These problems are chosen to represent a variety of possible eigenvalue distributions for the matrix A. In particular, problems 4–6 have eigenvalues on both sides of the imaginary axis and symmetric parts that have both positive and negative eigenvalues [KaSa86], features that make these problems especially challenging. Additionally, problem 4 has a condition number that grows rapidly with n; for $n = 32$ the condition number is already 6.0×10^{12}.

D.2. Stopping Conditions

Each experiment is terminated whenever the initial residual of the original system is reduced by 10^{-6}, that is, whenever

$$||r_k|| < 10^{-6}||r_0||, \tag{4.1}$$

where $r_k = f - Ax_k$. The only exception to this is when GMRES(k) is used with a preconditioner, in which case the residual of the preconditioned system is used. The outer CG iteration is started with an initial estimate of $x = 0$ so that the initial residual is simply the right hand side vector. An additional stopping condition is provided in case too many iterations of the CG algorithm are executed. This condition, however, is not met in any of the test runs.

D.3. Use of Parallelism

All tests are performed on an Alliant FX/8 at the Center for Supercomputing Research and Development of the University of Illinois. The FX/8 is a shared memory multiprocessor with 8 computational elements (CEs). Each CE has vector processing ability, with 8 vector registers containing 32 64–bit words each.

The speedup s_k realized by k CEs is defined as

$$s_k := t_1 / t_k,$$

where t_j is the time required for execution on j processors. In order to keep the other programming variables constant, for all runs Partition 1 is used and the least squares subproblems are solved by using a Cholesky factorization of the normal equations. Vectorization is used for both the 1 CE and the 8 CE runs.

Concurrency can occur in two basic ways in BSSOR. First, when computing a projection $y = P_i x$ each least squares subproblem can be solved simultaneously. For the partitioning used, this yields a possible n/4 parallel tasks. Vectorization can then be used within each least squares subproblem so that the overall projection is calculated in concurrent–outer/vector–inner mode. Second, the basic vector operations such as inner products and (vector + scalar*vector) required in the CG method can be spread among the available processors, i.e., executed in vector–concurrent mode. The latter is possible in any linear solver that involves long vectors, while the former is unique to BSSOR.

Speedups realized on the 8 CE's of the Alliant FX/8 range from 5 to 6 for problems tested.

D.4. Solving the Least Squares Subproblems

Before discussing the method of choice, some observations regarding the structure of C and C^TC are in order. For the above test problems, with the proposed partition (two block row partitioning) C is of the form,

$$C = \begin{bmatrix} D & 0 \\ T & D \\ D & T \\ 0 & D \end{bmatrix},$$

$$\text{and } C^TC = \begin{bmatrix} P & T \\ T^T & P \end{bmatrix}$$

In the above forms D, T, and P represent a diagonal, tridiagonal, and pentadiagonal matrix, respectively, where in general each one may be different.

From the minimax characterization of the singular values of C and A [GoVL83] it is easy to show that the condition number $\kappa(C)$ of C can never be worse than that of A. Table 4.4 shows the maximum of $\kappa(C)$ over all the subproblems involved in problem 4, along with the estimates for $\kappa(A)$ given in [KaSa86]. The condition of the subproblems is generally much better than the overall condition of A, and this holds for each of the 5 test problems. A possible explanation is that the ill–conditioning of A appears not in the subproblems but instead in the angles between the range spaces of P_1 and P_2.

Table 1: maxκ(C) vs. κ(A)

n	Condition Number	
	Max of $\kappa(C)$	$\kappa(A)$
8	0.2×10^2	0.8×10^2
16	0.2×10^2	0.2×10^6
32	0.1×10^2	0.6×10^{13}
64	0.1×10^2	*

* Not calculated

Another issue in choosing a solver for the least squares subproblems is that a stopping criterion must be set for the iterative methods. Let "inner" refer to the subproblem computations and "outer" refer to the computations specified by BSSOR. It is reasonable to require the inner accuracy to be at least as good as the outer accuracy. If it is much more accurate, then the projectors P_i are well defined and the number of outer iterations needed is the same as that required by the direct methods, but more work must be done on each inner iteration. Conversely if the inner accuracy required approaches that of the outer accuracy, less work is done on each inner iteration but now the projectors are defined with random errors on the order of that desired in the solution; more outer iterations will be performed and the algorithm may even fail to converge. Because one of the motivations for BSSOR is to achieve the robustness that other methods lack, all of the experiments use an inner tolerance for the residual norm that is 10 times more stringent than the outer tolerance.

Results comparing the solvers for problems 4 – 6 show that using preconditioned C.G. schemes for solving the least squares subproblems (normal equations) require the least time and storage. These methods have interesting behavior; as the outer iterations proceed the inner CG method requires fewer and fewer iterations.

D.5. Comparison with Other Methods

Four algorithms are compared to the BSSOR method. A version of PCGPAK supplies three: Generalized Conjugate Residuals (GCR(k)), Generalized Minimum Residual (GMRES(k)), and ORTHOMIN(k) [Elma82,SaSc86,EiES83]. The fourth is the conjugate gradient method used on the normal equations of A; this should not be confused with the use of conjugate gradients to solve the least square– subproblems in BSSOR. Preconditioners used in PCGPAK are ILU(0), MILU(0), and SSOR(1); while CG is tested with and without IC(0) preconditioning.

The stopping criterion adopted halts iterations when the relative residual norm is reduced by a factor of 10^{-6}. One important difference is that GMRES(k) must use the residual of the preconditioned system for the stopping criterion. Since the initial preconditioned residual can be many orders of magnitude larger than the unpreconditioned residual, this sometimes causes GMRES(k) to terminate prematurely. Another stopping criterion is necessary to guard against stalling, and in the experiments the iterations are terminated if the residual norm is not reduced by 10^{-6} within 10 iterations.

The version of PCGPAK used is optimized for the Alliant FX/8, and exploits both the vectorization and concurrent features of that machine [AnSa88].

CG is also used with the incomplete Cholesky factorization preconditioner IC(0). Since A^TA is not necessarily an M–matrix, there is no assurance that the factorization can be carried out to completion. A parameter α may be added to the diagonal elements during the factorization to allow it to proceed, but as with the PCGPAK methods this is set to 0 for the tests. It is important to note that in general such parameters are difficult to select adaptively, and in part the intent of these experiments is to use the methods as black–box solvers.

D.5.1. Problems Compared

Only a subset of problems 1–6 are presented here. Problem 3c has a positive definite symmetric part and so should be suitable for ORTHOMIN(k) and GCR(k). Problems 4–6 have indefinite symmetric parts; problem 4 additionally is ill–conditioned making it difficult for CG to solve. Problem size of n = 64 is used, so that full parallelism can be achieved by BSSOR.

D.5.2. Results

An upper limit of 2000 iterations is imposed on each run, and experiments that exceed that bound are regarded as failing and marked with an error code of MX. Three other conditions are labeled failures. If more than 10 iterations occur without a reduction in the residual the run is marked RS for "Residual Stalls". If the preconditioner cannot be formed, an error code of UP for "Unstable Preconditioner" is used. Finally, because GMRES(k) uses the residual for the preconditioned system as a stopping criterion, it can terminate iterations while the true residual is still large. If it exceeds 1, the run is labeled LR for "Large Residual".

Fig. 2 shows which failures occur for the various methods. Clearly BSSOR has a robustness unmatched by the other solvers.

Figure 3 compares the efficiency of BSSOR with the other unpreconditioned methods. The closest one in robustness is CG on the normal equations. The preconditioned methods, however, can out perform BSSOR when they work.

This comparison of BSSOR with preconditioned methods is unfair in one way. Only a default value of the parameter α defined in section 4.6.1.1 is used. In practice the user may be able to find values that prevent the factorization failures from occurring. This is not done here because the intent of the testing is to measure how the methods compare as "black–box" solvers. Given some knowledge of the systems being solved, as is the case when the same system is solved with several different right–hand sides, adaptively finding and using an optimal acceleration parameter may be more practical. Tables A.1 – A.4 summarize the results for problems 3C–6 with n = 64.

Another possible objection to the comparisons with the CG–like methods is that the value of k may be too small. To see if increasing k improves robustness, GMRES(k) is tested with k = 20 on problems 4 – 6 with n = 64.. The results are shown in Table A.5.

In summary, three results of the testing stand out. BSSOR is more reliable than the other solvers, is faster than the unpreconditioned methods, and is generally slower than the preconditioned methods when they succeed.

E. Conclusions and future research

The BSSOR method for solving linear systems is a remarkably robust iterative scheme that transforms a nonsymmetric system with an arbitrary eigenvalue distribution into a symmetric one with eigenvalues restricted to $[0,1)$. When applied to block tridiagonal systems concurrent operations on separate blocks becomes possible. Of the many block row partitionings that yield parallelism, the ones that allow the most concurrent tasks perform the best.

Figure 2: Failures of Methods

Method	Problem $n = 32$				$n = 64$			
	3c	4	5	6	3c	4	5	6
BSSOR								
GCR(3)		RS	RS			RS	RS	
GCR(3)-ILU		RS		MX		RS	RS	MX
GCR(3)-MILU				MX				UP
GCR(3)-SSOR	UP	MX		UP	UP	RS	RS	UP
GMRES(3)		RS	RS			RS	RS	
GMRES(3)-ILU		RS		LR		RS	RS	LR
GMRES(3)-MILU				LR				UP
GMRES(3)-SSOR	UP	RS		UP	UP	RS	RS	UP
ORTHOMIN(3)		RS	RS			RS	RS	
ORTHOMIN(3)-ILU		RS	RS	RS		RS	RS	RS
ORTHOMIN(3)-MILU			RS				RS	UP
ORTHOMIN(3)-SSOR	UP	RS	RS	UP	UP	RS	RS	UP
CG						MX		
CG-IC(0)		UP				UP		

Failure codes:
UP = Unstable Preconditioner
RS = Residual Stagnates
MX = Exceeds Maximum Iterations
LR = Large Residual

Figure 3: Times for Unpreconditioned Methods (sec)

n	Problem	Method				
		BSSOR	GCR(3)	GMRES(3)	ORTHOMIN(3)	CG
32	3c	0.94	8.80	8.74	5.41	1.85
	4	6.66	*	*	*	3.77
	5	2.43	*	*	*	2.09
	6	2.56	3.46	3.45	3.69	1.45
64	3c	5.99	50.30	49.22	49.31	14.35
	4	79.31	*	*	*	74.48
	5	28.42	*	*	*	43.05
	6	17.33	12.34	12.24	13.48	15.18

* Method Failed

For the problems examined here, each of the generated subproblems in turn is highly structured, making them amenable to being solved on a multiprocessor. This makes the algorithm particularly suited for a hierarchal memory machine such as Cedar. This multiprocessor is implemented with a global memory which supports up to 8 clusters; each cluster currently is an Alliant FX/8 with multiprocessing and vector capabilities and its own memory. With this architecture the separate blocks created by BSSOR can be assigned to individual clusters. Solving each least squares subproblem can then be done entirely

within a cluster. Global memory can hold the vectors defined by the outer CG iteration, and data communication between clusters can be exclusively through those vectors. Implementation of BSSOR on Cedar would allow the treatment of partial differential equations in three dimensions, where each least squares subproblem is large scale, sparse, and structured, and should be solved via an iterative scheme. This is an important future research issue for which this activity is in need of a more powerful multiprocessor, e.g. an FX/80 which has more powerful computational elements, larger cache, and larger memory.

BSSOR compares favorably with CG-like iterative methods for nonsymmetric systems when they are used without parameter estimation. The storage requirements are minimal and reliability is high. Usually BSSOR is more efficient than the unpreconditioned methods, and is slower only if the CG-like method is successfuliy used with a preconditioner.

Although testing has been restricted to block tridiagonal systems in this paper, BSSOR can be applied to other problems. The crucial property of block tridiagonal systems is that they can be easily row-permuted to yield separable systems in B_i^T. Any sparse system of equations that can be reordered into a banded system can also be solved using this method, only now the blocks have variable size. Narrower bandwidths will give greater parallelism and hence efficiency, but the method can still be applied even in the dense or sparse unstructured case [Kacz37]. Furthermore, the system need not be linear. These last two points, unstructured sparse linear problems and handling sparse nonlinear systems, are important research issues for further extension of BSSOR. Another potential extension of the method is to sparse least squares problems. The adjustment of geodetic networks, for example, leads to least squares problems where the coefficient matrix has block angular form [GoPS86]. This form requires no further permutation to be suitable for block row projection methods.

Table A.1: Comparison of Methods for Problem 3c, n = 64					
Method	Precond	Iters	Time	Residual	Failure Code
Block SSOR	-	116	0.599464E+01	0.575479E-03	
GCR(3)	-	1947	0.503018E+02	0.474242E-03	
	ILU	7	0.209307E+01	0.328218E-03	
	MILU	5	0.163364E+01	0.164215E-03	
	SSOR				UP
GMRES(3)	-	1947	0.492165E+02	0.474242E-03	
	ILU	7	0.189987E+01	0.328218E-03	
	MILU	5	0.151040E+01	0.164215E-03	
	SSOR				UP
ORTHOMIN(3)	-	1639	0.493057E+02	0.474713E-03	
	ILU	7	0.209663E+01	0.305439E-03	
	MILU	5	0.163631E+01	0.151166E-03	
	SSOR				UP
CG	-	616	0.143547E+02	0.471824E-03	
	IC(0)	7	0.944420E+00	0.893797E-04	
Failure codes:		UP = Unstable Preconditioner RS = Residual Stagnates MX = Exceeds Maximum Iterations LR = Large Residual			

Table A.2: Comparison of Methods for Problem 4, n = 64					
Method	Precond	Iters	Time	Residual	Failure Code
Block SSOR	-	300	0.793134E+02	0.479802E-04	
GCR(3)	-	629	0.161787E+02	0.150230E+02	RS
	ILU	46	0.994555E+01	0.315676E+02	RS
	MILU	1	0.851460E+00	0.181254E-06	
	SSOR	647	0.135435E+03	0.156203E+02	RS
GMRES(3)	-	629	0.162156E+02	0.150230E+02	RS
	ILU	38	0.745205E+01	0.315676E+02	RS
	MILU	1	0.801980E+00	0.181254E-06	
	SSOR	26	0.513346E+01	0.156204E+02	RS
ORTHOMIN(3)	-	17	0.528290E+00	0.176171E+02	RS
	ILU	224	0.466544E+02	0.325586E+02	RS
	MILU	1	0.847800E+00	0.181254E-06	
	SSOR	126	0.272351E+02	0.166143E+02	RS
CG	-	2000	0.429486E+02	0.944334E-03	MX
	IC(0)				UP
Failure codes:		UP = Unstable Preconditioner RS = Residual Stagnates MX = Exceeds Maximum Iterations LR = Large Residual			

Table A.3: Comparison of Methods for Problem 5, n = 64					
Method	Precond	Iters	Time	Residual	Failure Code
Block SSOR	–	300	0.284214E+02	0.975404E–05	
GCR(3)	–	133	0.344119E+01	0.605459E+00	RS
	ILU	114	0.236077E+02	0.477515E+00	RS
	MILU	1	0.893210E+00	0.561537E–12	
	SSOR	189	0.397837E+02	0.433020E+00	RS
GMRES(3)	–	133	0.347982E+01	0.605459E+00	RS
	ILU	70	0.131598E+02	0.477517E+00	RS
	MILU	1	0.892620E+00	0.560116E–12	
	SSOR	118	0.224104E+02	0.433020E+00	RS
ORTHOMIN(3)	–	65	0.196827E+01	0.613196E+00	RS
	ILU	96	0.203637E+02	0.528433E+00	RS
	MILU	1	0.889790E+00	0.561537E–12	
	SSOR	76	0.166553E+02	0.479316E+00	RS
CG	–	1864	0.430476E+02	0.148442E–04	
	IC(0)	396	0.352268E+02	0.150832E–04	
Failure codes:	UP = Unstable Preconditioner RS = Residual Stagnates MX = Exceeds Maximum Iterations LR = Large Residual				

Table A.4: Comparison of Methods for Problem 6, n = 64					
Method	Precond	Iters	Time	Residual	Failure Code
Block SSOR	–	109	0.173253E+02	0.192833E–03	
GCR(3)	–	483	0.123350E+02	0.254707E–03	
	ILU	2000	0.401148E+03	0.149972E+06	MX
	MILU				UP
	SSOR				UP
GMRES(3)	–	483	0.122430E+02	0.254707E–03	
	ILU	2	0.111076E+01	0.285255E+03	LR
	MILU				UP
	SSOR				UP
ORTHOMIN(3)	–	458	0.134819E+02	0.240795E–03	
	ILU	206	0.428312E+02	0.149968E+06	RS
	MILU				UP
	SSOR				UP
CG	–	657	0.151798E+02	0.256340E–03	
	IC(0)	90	0.855858E+01	0.252914E–03	
Failure codes:	UP = Unstable Preconditioner RS = Residual Stagnates MX = Exceeds Maximum Iterations LR = Large Residual				

Table A.5: GMRES(20) with n = 64					
Problem	Precond	Iters	Time	Residual	Notes
4	–	77	0.279077E+01	0.127500E+02	RS
	ILU	52	0.105483E+02	0.212383E+02	RS
	MILU	1	0.804430E+00	0.181254E–06	
	SSOR	31	0.631055E+01	0.158618E+02	RS
5	–	218	0.787123E+01	0.497596E+00	RS
	ILU	79	0.158311E+02	0.730619E–05	
	MILU	1	0.884910E+00	0.560116E–12	
	SSOR	103	0.208793E+02	0.905222E–05	
6	–	476	0.172964E+02	0.249851E–03	
	ILU	2	0.112231E+01	0.265255E+03	LR
	MILU	–	–	–	UP
	SSOR	–	–	–	UP
Failure codes:		UP = Unstable Preconditioner RS = Residual Stagnates MAXIT = Exceeds Maximum Iterations LR = Large Residual			

1. References

[AnSa88]
E. Anderson and Y. Saad, "Preconditioned conjugate gradient methods for general sparse matrices on shared memory machines", to appear in the proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing, Los Angeles, December 1987.

[BjEl79]
A. Bjorck and T. Elfving, "Accelerated projection methods for computing pseudo–inverse solutions of systems of linear equations," *BIT*, vol. 19, pp. 145–163, 1979.

[BrSa88]
R. Bramley and A. Sameh, "A Robust Parallel Solver for Block tridiagonal systems," Proceedings 1988 International Conference on Supercomputing, pp. 39–54.

[DBMS79]
J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *Linpack User's Guide*, SIAM, Philadelphia, 1979.

[Doga85]
J.J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment," ANL MCS–TM–23, July 1985.

[EiES83]
S. Eisenstat, H. Elman, and M. Schultz, "Variational iterative methods for nonsymmetric systems of linear equations", *SIAM J. Numerical Analysis*, vol. 20, pp. 345–357, 1983.

[Elfv78]
T. Elfving, "On the conjugate gradient method for solving linear least squares problems", LiTH–MAT–R–1978–3, Department of Mathematics, Linkoping University, S–58183 Linkoping Sweden, 1978.

[Elma82]
H. Elman, "Iterative methods for large, sparse, nonsymmetric systems of linear equations", Yale University Res. Rept. 229, Princeton, April 1982.

[FHHJ83]
J. Fromm, et al., "Experience with Performance Measurement and Modeling of a Processor Array," IEEE Trans. on Comp., Vol. C–32, No.1, Jan. 1983, pp. 15–31.

[Gabr85]
Gabriel, R.P., Performance and Evaluation of Lisp Systems, MIT Press, 1985.

[GJMY87]
K. Gallivan, et al., "Performance Analysis on the Cedar System," to appear in *Performance Evaluatioin of Supercomputers*, J.L. Martin Ed., North–Holland, 1988.

[GoPS86]
G. Golub, R. Plemmons and A. Sameh, "Parallel block schemes for large scale least squares computations", CSRD Rept. 574, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, April 1986.

[GoVL83]
G. Golub and C. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, 1983.

[HaPa88]
Harrison, L. and Padua, D. PARCEL: Project for the Automatic Restructuring and Concurrent Evaluation of Lisp, Proceedings of the 1988 ACM International Conference on Supercomputing, ACM, July 1988.

[Kacz37]
S. Kaczmarz, "Angenaherte Auflosung von Systemen linearer Gleichungen," *Bull. Internat. Polon.*

Sci. Cl. A, pp. 355–357, 1937.

[KaSa86]
C. Kamath and A. Sameh, "A projection method for solving nonsymmetric linear systems on multiprocessors," CSRD Rept. 611, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, October 1986.

[KDLS86]
D.J. Kuck, et al., "Parallel Supercomputing Today and Cedar Approach," Science, Feb. 1986, pp. 967–974.

[KuSa87]
D.J. Kuck and A. Sameh, "A Supercomputing Performance Evaluation Plan," to appear in Proc. of Int'l Conf. on Supercomputing, 1987.

[McMa86]
F.H. McMahon, "The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range," Lawrence Livermore National Lab., UCRL–53745, Dec. 1986.

[PaSa82]
C. Paige and M. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. on Math. Soft.,* vol. 8, no. 1, pp. 43–71, March 1982.

[PfNo85]
G.F. Pfister and V.A. Norton, "Hot–Spot Contention and Combining in Multistage Interconnection Networks," IEEE Trans. on Comp., Vol. C–34, No.10, Oct. 1985, pp. 943–948.

[Reid71]
J. Reid, "On the method of conjugate gradients for the solution of large sparse systems of linear equations," *Large Sparse Sets of Linear Equations,* ed. J. Reid, Academic Press, pp. 231–254, 1971.

[SaSc86]
Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.,* vol. 7, no. 3, pp. 856–869, July 1986.

[SeSS83]
Z. Segal, et al., "An Integrated Instrumentation Environment for Multiprocessors," IEEE Trans. on Computers, Vol. C–32, No. 1, Jan. 1983.

[SiWa87]
M.L. Simmons and H.J. Wasserman, "Los Almos National Laboratory Computer Benchmarking 1986," Los Alamos National Lab., LA–10898–MS, Jan. 1987.

[TuVe87]
S. Turner and A. Veidenbaum, "Register–Transfer Level Parallel Simulator for Interconnection Networks/Shared Memory System," Internal memo, U. of Illinois at Urbana–Champaign, CSRD, Nov. 1987.